The Shape of Light:
Understanding GPI PSF Distributions

Beck Dacus
2021 Physics REU, Department of Physics, University of California, Santa Barbara and Department of Physics and Astronomy, Amherst College

Maxwell Millar-Blanchaer (Faculty Advisor)
Department of Physics, University of California, Santa Barbara
(Dated: September 30, 2021)

The Gemini South Telescope in Chile, with its Gemini Planet Imager (GPI) instrument, is a pioneering facility in the field of "direct imaging"— its integral field polarimeter can produce images of protoplanetary and debris disks with light from the host star almost completely suppressed. To do this, an array of lenslets divides the image into several thousand "spots," which must then be sorted by polarization and converted into single pixels one-by-one. Some simplifying assumptions make this process more tractable, and published results indicate that these assumptions do not compromise the data; however, this does not mean they cannot be improved upon. In this paper, we test the assumption that each spot's point-spread function (PSF) on the detector is a two-dimensional Gaussian, and suggest alternatives to traditional aperture photometry in an attempt to improve the signal-to-noise ratio (SNR) of extracted images. We find that sampling sections of a raw image to empirically model spots replicates individual spot features substantially better than functional-form profiles, which could be useful when implementing a weighting scheme during flux extraction.

I. INTRODUCTION

A. Polarimetry

A number of devices make the Gemini Planet Imager (GPI) one of the most powerful direct imaging instruments in the world, not the least of which is its adaptive optics subsystem, allowing it to rival the angular resolution of the largest space telescopes. However, the focus of this paper is the integral field polarimeter (IFP) (see Figure 1). GPI's IFP first passes incoming light through a lenslet array, splitting the image into ~ 36000 optical paths. Each of these then passes through a polarizing beamsplitter, separating the light from that point on the image into its two orthogonally polarized components. These pairs of spots then fall onto the detector; the data reduction pipeline (DRP) is responsible for extracting spots of each polarization into their own images (see Section I B).

The two resulting images look very similar, in that they are dominated by light from the central star. In fact, because starlight should be unpolarized, the two images of the star should be almost identical, even given the optical abberations of the telescope and atmosphere, since both images were taken simultaneously. However, if a disk of dust particles is orbiting the star, it will preferentially polarize any light it reflects. While this disk will be much fainter than the star it orbits, each point on the disk will not be the same brightness in both images. This means that if we subtract one of these images from the other, the star should disappear almost completely, while any disk surrounding it should be clearly visible. This method of starlight subtraction does not depend on angular or spectral diversity, and is therefore an incredibly powerful tool for extracting images of disks with minimal systematic error.

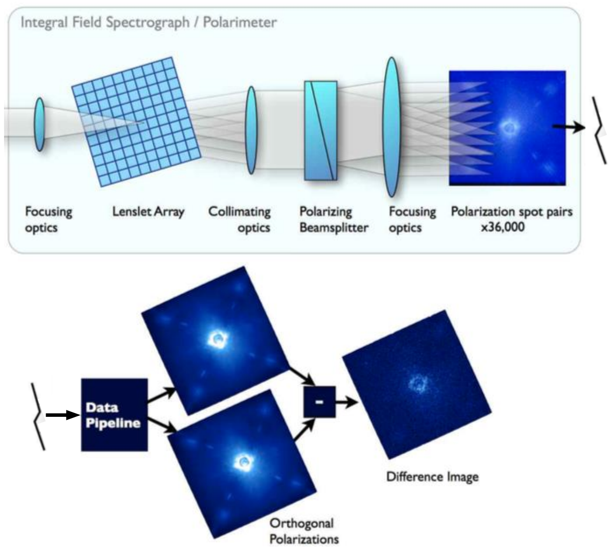

FIG. 1. A flowchart of light's path through the Gemini Planet Imager's integral field polarimeter, and the resulting data's path through the data reduction pipeline. Adapted from [1].

B. GPI Pipeline

To turn the raw image of overlapping polarization states into separate pictures, we must first identify the positions and polarizations of each spot on the raw image. This is done through iterative solving in the GPI data reduction pipeline (DRP). Starting from a set of initial guesses for each spot's coordinates, the pipeline fits a two-dimensional Gaussian profile to find the true, sub-pixel center of each point-spread function (PSF). The pipeline then aligns a pre-saved reference grid to the centers of those spots, which tells it each spot's polarization. It then performs a modified version of aperture photometry on each spot. In regular aperture photometry, one obtains a star's "flux" by defining an area around its center in which the pixels deviate significantly from the background, then subtracting out that background and summing the remaining pixel values in that area. While this has worked very well for measuring the brightnesses of *stars* in astronomical images, the principle does not quite apply in this case: the polarimetric spots do not represent a physical point source, but all the light that falls onto an individual lenslet, including sources of noise like sky background. It then makes sense to account for the characteristic shape of these spots by weighting each pixel's value in the sum; pixels further from the center of a spot will be dimmer, and thus more affected by detector noise. If we weight these pixels less than those at the centers of the spots, the noise in these areas (very little of which comes from the sky itself) will have less influence on the final image, and should improve signal-to-noise ratios (SNRs) in reduced GPI data.

An elegantly simple solution might use the Gaussian fit parameters we obtained when fitting for the spot center positions, normalizing those profiles to create a tailored weighting function for each spot. However, extracting images this way was actually found to *decrease* the SNR. The Gaussian weighting profile that maximized SNR was three times wider than GPI's best fit; this indicated that a 2-dimensional Gaussian may not be the best weighting function for GPI's spots. In this paper we set out to determine the precise shape of these spots, with the hope that this would point to an optimal weighting function for the PSFs.

II. METHODS

To test the efficacy of each model, we first used the GPI DRP to obtain the center positions for the spots in our raw image. We then generated multiple artificial GPI images, where each one assumed the spot shapes followed a different distribution.

A. Generating Calibration File

A pre-compiled version of the GPI DRP is publicly available online[1], precluding the need for an IDL license in this work; this is the software we used to create our spot positions calibration file. We sourced our raw data files from the polarimetry data reduction tutorial[2], specifically the "lamp" files therein. While the lenslet array makes it impractical to obtain true flat-field calibration images for GPI, these lamp files offer something analogous, ensuring that most spots are of approximately the same brightness ([2]). We ran the recipe "Calibrate Polarization Spot Locations" on these files to obtain their fit parameters— see Appendix A for details.

B. Creating Artificial Images

With all calibration files formatted as FITS images, we read them into a Python Jupyter Notebook using the `astropy.io.fits` package. We fit each model to a 6-by-6 pixel cutout image of each spot, and evaluated them in a box of the same size. This avoided contamination from surrounding spots when fitting our models.

1. GPI Gaussian Fit Parameters

Along with sub-pixel center positions, the GPI DRP fits Gaussian 2% radii (the distance at which the spot reaches 2% of its peak central value, in pixels) in the x and y directions, as well as the rotation angle of the PSF. It does not fit peak height values for the PSFs; we determined this value ourselves in two ways. The first method involved generating an artificial image where all the peaks were set to 1, then dividing the original image by this scaled-down artificial version. We then isolated a 3-by-3 pixel box around the pixel closest to the PSF center and took the median of all the pixel values therein. This gave us an estimate for the factor by which each pixel in the two images differed, which should be roughly constant if they are the same shape. We then generated these same height-normalized Gaussians, this time passing them into the `scipy.optimize.minimize` function to minimize the residuals by (1) multiplying each pixel value by a constant factor (the Gaussian's peak height), and (2) adding a fixed constant to every pixel (a smooth background). For our initial best guesses for each parameter, we used (1) the result of the previous peak height scaling method and (2) the approximate mode of the pixel values when they are represented as a histogram (Figure 2), respectively.

2. Astropy Gaussian Fit

In addition, we fit our own Gaussians to each spot using the `astropy.modeling` package as an independent

[1] `http://docs.planetimager.org/pipeline/installation/install_compiled.htmlinstalling-from-compiled`

[2] `http://docs.planetimager.org/pipeline/usage/tutorial_pol.htmlusage-quickstart-pol`

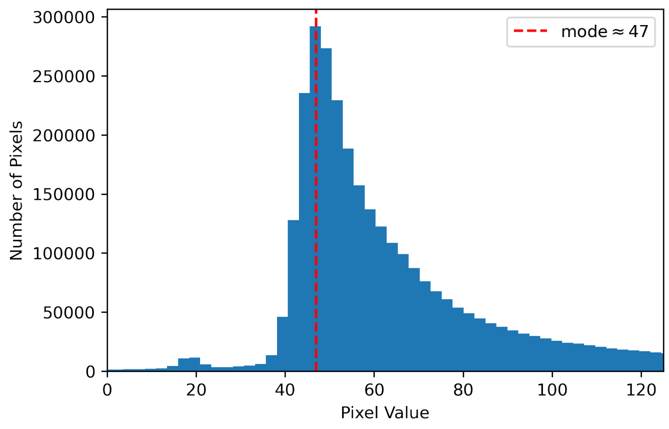

FIG. 2. Histogram showing the number of pixels with a given pixel value; the average background level of the whole image is assumed to be the mode of this distribution, highlighted with the broken red line.

test of the Gaussian profile, in case there were peculiarities in the GPI DRP fitting process. GPI fit parameters and the results of our best-fit peak height values were used as initial guesses in each model, which were then optimized. We combined each of these individual Gaussians with a `Const2D` background fit, with the results of the `scipy.optimize.minimize` background fit as the initial best guess.

3. *Moffat Profile*

Because the GPI DRP only fits Gaussians to raw image spots, we performed our own `astropy.modeling` fitting for the Moffat distribution. The default `Moffat2D` fitting function in this package cannot elongate or rotate the profiles in any dimension; this functionality is necessary, since the PSFs at the edge of the image are highly tilted and elongated by system optics. Fortunately, this package allows the user to create a custom model, which it can then optimize for individual spots. Our custom Moffat profile model is very similar to the default, only adjusting for rotation angle and widths in each direction; the code for this function is included in Appendix B. Similar to the Gaussian fit, we added a `Const2D` model to replicate the background noise.

4. *Effective Point-Spread Function (ePSF)*

After analyzing the efficacy of two well-studied distributions with known functional forms, we decided to take the approach outlined in [3] to make an empirical model specific to the spots in GPI images. We did this using the Python package `photutils.psf`; for each spot we wanted to model, we identified a sample of surrounding spots from the original image and extracted cutouts of

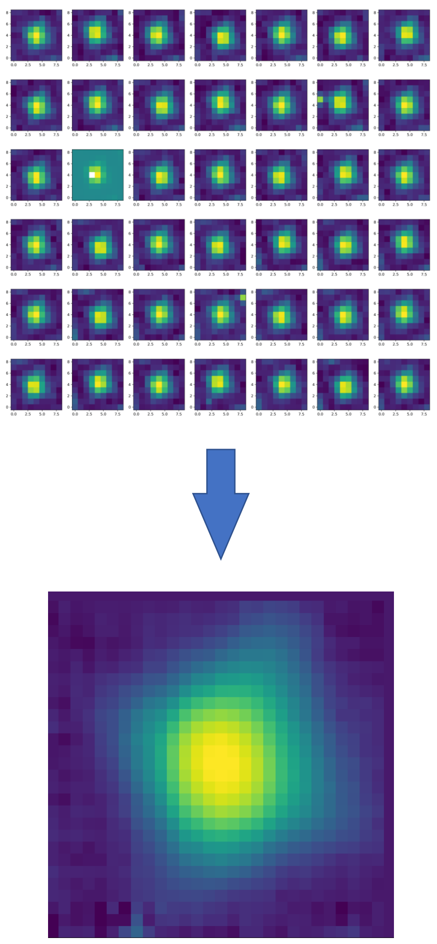

FIG. 3. An example of an effective PSF model constructed from ~ 42 lamp image PSFs, with relative brightness plotted logarithmically.

them with the `extract_stars` function. We then passed that sample to an `EPSFBuilder` object that constructed an empirical model based on those spots (see Figure 3). The spot we intended to model was never included in the sample.

When calling the `extract_stars` function, the x and y coordinates of each spot must be passed in; the number of spots included in each model is therefore a user-input parameter. While including too many spots in each model can lead to impractical runtimes, it can also *worsen* errors beyond a certain threshold. A spot's position on the detector has a nontrivial effect on the shape of its PSF; specifically, spots are elongated along the line that connects them to the center of the image, an effect that grows with distance from the center. This means that drawing on spots too far away from the target spot may systematically alter the shape of the resulting model, making it a worse fit. Our findings for the optimum number of spots to include are discussed further in Section IV.

Further, generating a single model can take anywhere from a few seconds to several minutes; in any case, doing so for each of the $\sim$ 72000 spots in the image on a single processor would be far too time-consuming. For this reason, we did not want to generate a separate model for every individual spot; instead, we wanted to "recycle" a single model on multiple spots, as long as they had similar errors. To determine the optimal number of spots to recycle over, we subtracted a single model from all spots in a 100-by-100 pixel box (see Figure 4) and plotted the root mean square residuals (see Section III) as a function of distance from the central spot. The results of this are outlined in Section IV.

III. ANALYSIS

For each method of artificial image generation, we subtracted the artificial image from the original to obtain a two-dimensional map of our residuals (see Figures 4 and 5). We then looked at the area surrounding each individual PSF and calculated the root-mean square (RMS) of their residuals, following Equation 1:

$$\mathrm{RMS} = \sqrt{\bar{x^2}} \tag{1}$$

where x^2 represents the collection of residual pixel values, squared element-wise. We then directly compared the RMS distributions of each method using histograms of individual PSF RMS values (Figure 6).

IV. RESULTS

Out of all the methods we tested, the GPI DRP's Gaussian fit performed worst— our own Gaussians were only marginally better. Moffat profiles were a substantial improvement, but ePSF models performed best by a significant margin, even before we optimized the number of models generated, or the number of spots included in each model; the top histogram in Figure 6 therefore represents an upper limit.

With this knowledge, we began trying to optimize the ePSF models. First we tested the optimum number of spots to incorporate when generating each model— see Section II B 4 for an explanation of why including too many spots can increase errors. At multiple places in the image, we identified a "central spot" that we wished to model, then defined square regions of increasing size centered on that spot. For each such box, we made a model incorporating all of the spots (excluding the central one) whose centers were bounded by that box. Each model was then subtracted from the central spot; Figure 7 shows the results of three such investigations, displaying a characteristic error minimum when $\sim$ 170 surrounding spots are used. Similar trends were observed at other sites across the image, leading us to adopt this number as the optimal model-generating sample size.

We then investigated the relationship between RMS residuals and distance when using a single model, to see how many times we might be able to recycle each model when generating an artificial image. Assuming that generating each ePSF model represents the majority of overall runtime, the factor by which this runtime is reduced is equal to the number of spots we recycle one model on; for instance, using the same model on the 3 centermost spots could generate an artificial image 3 times faster than one requiring one model per spot. We selected a distance cutoff based on where the highest RMS residuals begin to approach the spread of the RMS-vs-distance relation in Figure 8. Based on our plot, we set this distance to be

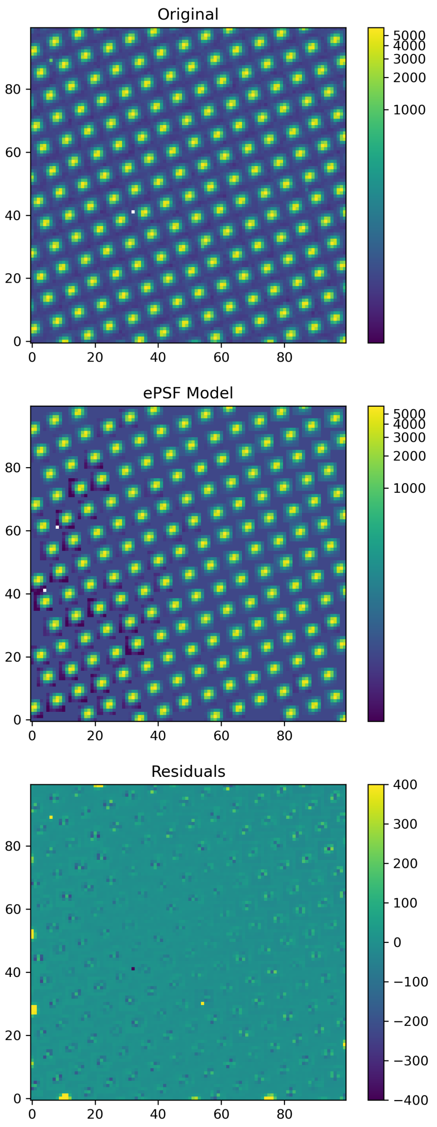

FIG. 4. *Top:* A subsection of the original flat-field image, on a logarithmic color scale. *Center:* An artificial ePSF image, using a single model (plus a local background fit) to replicate each spot in the same region of the original flat field. *Bottom:* An image where each pixel represents the difference of the pixels in the two other images at the same location. Variation in the residuals increases with distance from the center, indicating that the model is less effective in these regions.

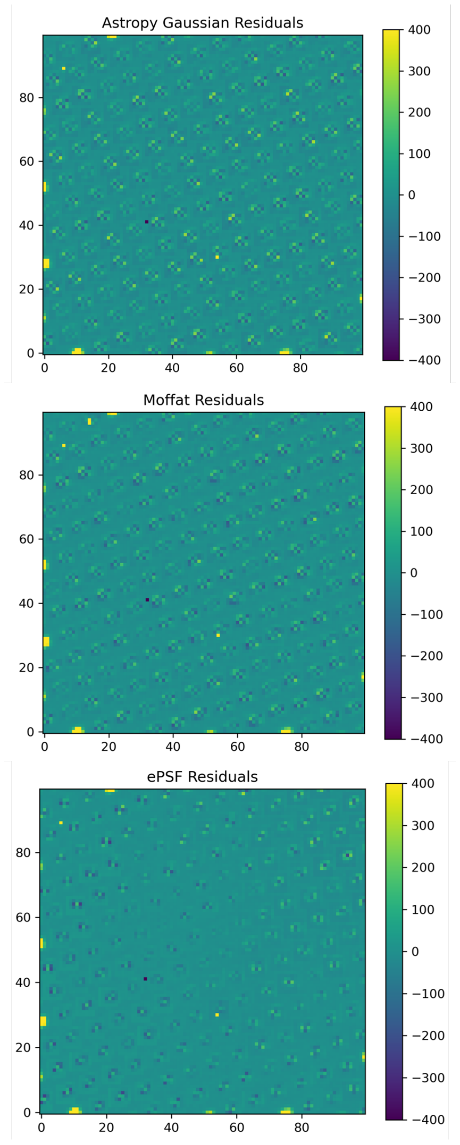

FIG. 5. Three residual images, the results of creating artificial model images (from top to bottom: Gaussian, Moffat, ePSF) and subtracting them from the original image.

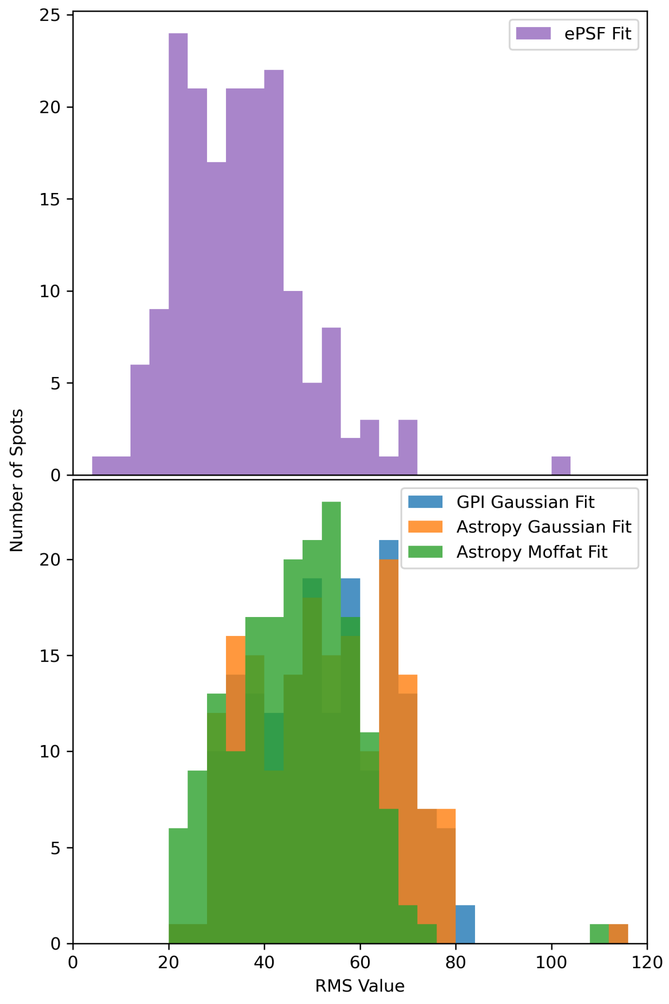

FIG. 6. *Top:* A histogram of the residual root-mean squares for the ePSF fits in a 100-by-100 pixel section of the image. *Bottom:* An RMS residual histogram for our three functional-form models (GPI Gaussian, Astropy Gaussian, and Moffat) in the same part of the image.

10 pixels, which would lead us to recycle a single model over the 5 to 7 centermost spots. This could make image generation 5 to 7 times faster, even without running it on a faster computer and/or on multiple processors.

V. FUTURE WORK

The ultimate test of our ePSF models will be to generate an artificial version of our calibration image before running that through the same GPI DRP recipe as the original image; specifically, we want to see if this results in the same extracted flux as the original. However, we will first need to limit the runtime of the code that generates the full artificial image, not only to speed up our investigation but to make this process practical if it is ever incorporated into the DRP itself. We hope to do this by recycling models on multiple spots (see Figure 8), and by executing our code on several processing cores running in parallel.

ACKNOWLEDGMENTS

I would first like to acknowledge my advisor and principal investigator, Professor Maxwell Millar-Blanchaer, for guiding me through this project and for making the time to work around my remote schedule. I also want to thank my graduate mentor Skyler Palatnick for providing further support and helping me with Python issues. Instrumental to my whole experience was Dr. Sathya Guruswamy, who directed the REU program and offered

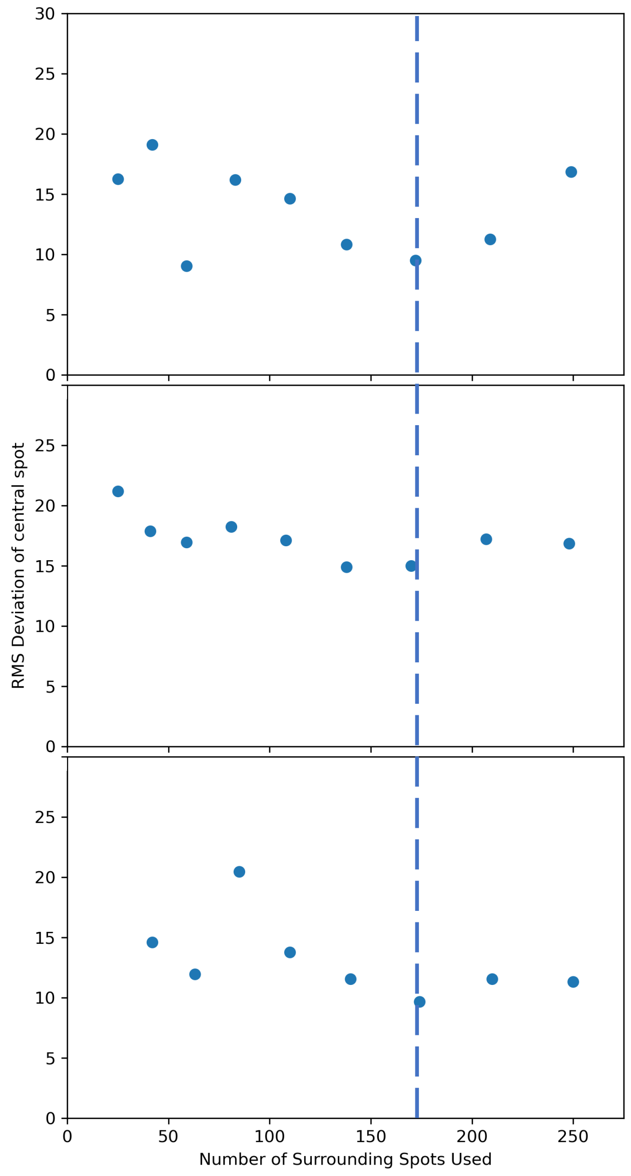

FIG. 7. The RMS of the residuals between a spot and its corresponding model, plotted as a function of the number of surrounding spots used to generate said model, for three parts of our image. While RMS values can vary wildly when very few spots are incorporated into the model, there is a distinct "saddle" shape at higher numbers, with its lowest point at ~ 170 spots.

constant support throughout the summer. This research was funded by the National Science Foundation, under grant number PHY-1852574.

Appendix A: GPI DRP Recipe

While the GPI DRP is an IDL program, this project made use of the pre-compiled version and thus only interfaced directly with the GUI, recipe editor, and status console. To generate our desired calibration file, we downloaded arc lamp exposures from GPI's "Polarimetry data reduction" tutorial dataset and broadly followed the instructions therein to obtain spot positions and Gaussian fit parameters for the file "`S20131212S0022.fits`". However, this tutorial assumed the reader had an IDL license, and thus full access to all of the recipes within the pipeline— this meant that, rather than using the the "Calibrate Polarization Spot Locations - Parallel" recipe, we used the unparallelized version. No primitives or parameters were otherwise altered.

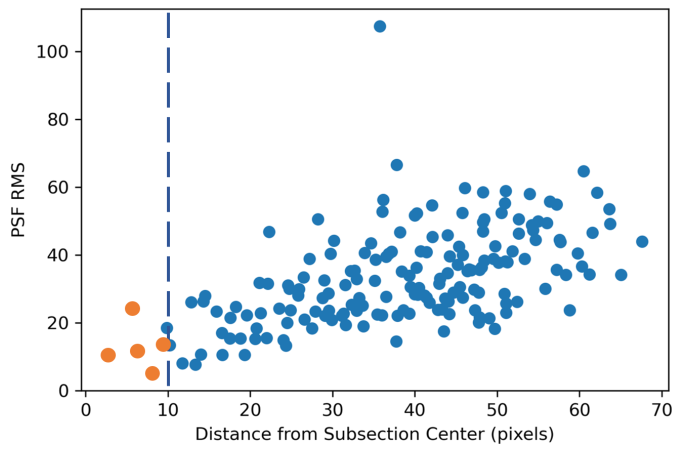

FIG. 8. A plot of the RMS deviation between a model and spots at varying distances from the central spot. The apparent RMS "floor," which trends linearly with distance, seems to intersect the horizontal axis at around 10 pixels; recycling a single model on even the five-or-so spots within this range could result in a five-fold reduction of the program's runtime without significantly increasing errors.

Appendix B: Python Notebook

FITS data from the GPI instrument and DRP were read into the Jupyter Notebook development environment and manipulated using Python 3, assisted by several mathematical, modeling, and plotting packages:

- `numpy` (for array manipulation, arithmetic, and statistics)
- `matplotlib.pyplot` (for data visualizations)
- `astropy.io.fits` (to manipulate `.fits` files)
- `astropy.stats` (for more statistics)
- `astropy.stats.sigma_clip` (for rejecting bad pixels)
- `astropy.modeling.models` (for generating and evaluating PSF models)
- `astropy.modeling.fitting` (for optimizing models to match PSFs)

- `scipy.optimize.minimize` (to exert control over which/how parameters are fit)
- `photutils.psf.extract_stars` (to easily make cutouts of stars in a format compatible with `EPSFBuilder`)
- `photutils.psf.EPSFBuilder` (to generate ePSF models based on training data)

In several cases, we optimized models using the `scipy.optimize.minimize()` function; however, one of its arguments is a function which returns a floating point number. `minimize()` minimizes this number by varying the values passed in as the first argument, which must be an iterable object if multiple parameters are being fit. To optimize the peak and background level in the Gaussian fits which otherwise used GPI parameters, we passed in this function:

```
def total_residuals(x,data,model):
   difference = data - ((x[0]*model)+x[1])
   return np.sum(difference**2)
```

By minimizing the value returned, `minimize()` brought the residuals as close to zero as it could, resulting in the optimum peak height (`x[0]`) and background (`x[1]`) with the parameters given.

Similarly, when subtracting our ePSF, we had to optimize on GPI's estimates for the center of the spot, as well as the "flux" (the sum of all the pixel values in the cutout box) and background level. For this, we passed in the following function:

```
def epsf_residuals(guesses, x,y,data,model):
   difference = data -
   (model.evaluate(x=x,y=y,
                   flux=guesses[0],
                   x_0=guesses[1],
                   y_0=guesses[2])
    +guesses[3])

    return np.sum(difference**2)
```

We also created a custom two-dimensional Moffat profile function, which `astropy.modeling.models` then used to fit our improved Moffat profiles to our spots. This function is given below:

```
def Moffat2D(x, y, amplitude=1.0,
             x_0=0.0, y_0=0.0,
             x_gamma=1.0, y_gamma=1.0,
             theta=0.0, alpha=1.0):

    theta_rads = (theta/360)*2*np.pi

    x_new = (x*np.cos(theta_rads)) -
            (y*np.sin(theta_rads))
    y_new = (x*np.sin(theta_rads)) +
            (y*np.cos(theta_rads))

    rr_gg = (((x_new - x_0)/x_gamma) ** 2 +
            ((y_new - y_0)/y_gamma) ** 2)

    return amplitude * (1 + rr_gg) ** (-alpha)
```

[1] M. D. Perrin, G. Duchene, M. Millar-Blanchaer, M. P. Fitzgerald, J. R. Graham, S. J. Wiktorowicz, P. G. Kalas, B. Macintosh, B. Bauman, A. Cardwell, J. Chilcote, R. J. De Rosa, D. Dillon, R. Doyon, J. Dunn, D. Erikson, D. Gavel, S. Goodsell, M. Hartung, P. Hibon, P. Ingraham, D. Kerley, Q. Konapacky, J. E. Larkin, J. Maire, F. Marchis, C. Marois, T. Mittal, K. M. Morzinski, B. R. Oppenheimer, D. W. Palmer, J. Patience, L. Poyneer, L. Pueyo, F. T. Rantakyrö, N. Sadakuni, L. Saddlemyer, D. Savransky, R. Soummer, A. Sivaramakrishnan, I. Song, S. Thomas, J. K. Wallace, J. J. Wang, and S. G. Wolff, Polarimetry with the Gemini Planet Imager: Methods, Performance at First Light, and the Circumstellar Ring around HR 4796A, Astrophys. J. **799**, 182 (2015), arXiv:1407.2495 [astro-ph.EP].

[2] P. Ingraham, M. D. Perrin, N. Sadakuni, J.-B. Ruffio, J. Maire, J. Chilcote, J. Larkin, F. Marchis, R. Galicher, and J. Weiss, Gemini planet imager observational calibrations II: detector performance and calibration, in *Ground-based and Airborne Instrumentation for Astronomy V*, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 9147, edited by S. K. Ramsay, I. S. McLean, and H. Takami (2014) p. 91477O, arXiv:1407.2302 [astro-ph.IM].

[3] P. Ingraham, J.-B. Ruffio, M. D. Perrin, S. G. Wolff, Z. H. Draper, J. Maire, F. Marchis, and V. Fesquet, Gemini planet imager observational calibrations III: empirical measurement methods and applications of high-resolution microlens PSFs, in *Ground-based and Airborne Instrumentation for Astronomy V*, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 9147, edited by S. K. Ramsay, I. S. McLean, and H. Takami (2014) p. 91477K, arXiv:1407.2303 [astro-ph.IM].